IN THE CLAIMS

1. (currently amended) A process for developing mathematically validated object oriented software comprising the steps of:

a) writing an abstract specification of a class, methods and expected properties of a component of the software, wherein the abstract specification of the methods includes a two-part postcondition such that when one of the methods is overridden by an overriding method in a descendent class one part of the one of the methods is inherited and another part the one of the methods is not inherited by the overriding method;

b) checking the abstract specification for errors and verifying that the class has the expected properties;

c) generating executable code for the class from the abstract specification;

d) running and evaluating the executable code to check that the code meets requirements other than a required speed of performance; and

e) evaluating the speed of performance when handling data sets commensurate to the size of data sets the software component is required to handle.

2. (original) A process as claimed in claim 1, wherein step c) of generating executable code includes the further step, where the specification is too complex to generate executable code directly, of refining the method by specifying a series of operations to be undertaken, to produce a refined method and verifying that the refined method behaves in accordance with the abstract specification before generating executable code.

3. (original) A process as claimed in claim 1, wherein step d) of running and evaluating the executable code, includes the further step, where the code does not meet requirements, of identifying the defects and correcting the abstract specification and then repeating the process from step b).

4. (original) A process as claimed in claim 1, wherein the step e) of evaluating the speed of performance, includes the further steps, where the speed of performance is inadequate, of restructuring the data maintained by the class, adding variable declarations for the restructured data, and refining the methods and constructors to take account of the restructured data and generating executable code.

5. (original) A process as claimed in claim 1, wherein the step e) of evaluating the speed of performance, includes the further steps, where the speed of performance is inadequate, of refining the method to produce a refined method and verifying that the refined method meets the specifications of the original method, and generating executable code.

6. (original) A process as claimed in claim 1, wherein the abstract specification includes a name of the class, a list of other class or classes and/or interfaces that said class inherits from, an abstract model of data maintained by the class, abstract specifications of the methods and constructors of the class, and theorems of expected behaviour of the class and the methods and constructors.

7. (original) A process as claimed in claim 6, wherein the abstract model of data includes declarations of abstract variables and may optionally include class invariants which are conditions concerning values of the abstract variables that are expected always to be true and declarations of abstract methods private to the class that assist in defining the class invariants and what other methods achieve.

8. (original) A process as claimed in claim 6, wherein the abstract specifications of the methods and constructors of the class include: a method name; a method parameter list; and a definition of what the method achieves.

9. (original) A process as claimed in claim 8, wherein the abstract specifications of the methods and constructors of the class further include one or more of: a method result type; a precondition that is required to hold whenever the method is called; a post-assertion description of conditions expected to hold when the method returns.

10. (original) A process as claimed in claim 2, wherein the step of refining the method includes providing an algorithm of program statements including postcondition statements.

11. (original) A process as claimed in claim 6, wherein a description of the class is divided into regions: an abstract region containing the abstract data model and also private methods and constructors referred to elsewhere in specifications of the class; an internal region containing re-implemented data and redundant data and also private methods and

constructors referred to in re-implementations of other methods; a confined region of methods and constructors that are used only within the class and within other classes that inherit from that class; and an interface region of methods and constructors accessible to any program or components that use instances of the class.

12. (original) A process as claimed in claim 1, wherein the step c) of generating executable code includes the steps of:

i) tokenizing the abstract specification to form a token stream and building a representation of the abstract specification;

ii) passing the token stream;

iii) binding identifiers and operators to declarations;

iv) converting the specifications contained therein to standard forms;

v) generating a new instance of each variable at every point at which the variable is changed, effectively replacing each variable by a succession of constants;

vi) generating proof obligations to represent requirements for program correctness;

vii) proving the obligations;

viii) using the abstract specification to generate a test harness or a set of test data for testing speed of performance;

ix) generating code statements to implement the specification where no code is provided;

x) translating the code statements to easily translatable form; and

xi) translating the easily translatable form to an output language.

13. (canceled)

14. (currently amended) A system for developing mathematically validated object oriented software comprising:

a) means for writing an abstract specification of ~~the~~ a class, methods and expected properties of a component of the software ~~component~~, wherein the abstract specification of the methods includes a two-part postcondition such that when one of the methods is overridden by an overriding method in a descendent class one part of the one of the methods is inherited and another part the one of the methods is not inherited by the overriding method;

b) means for checking the abstract specification for errors and verifying that the class has the expected properties;

c) means for generating executable code for the class from the abstract specification;

d) means for running and evaluating the executable code to check that the code meets requirements other than a required speed of performance; and

e) means for evaluating the speed of performance when handling data sets commensurate to the size of data sets the software component is required to handle.

15. (original) A system as claimed in claim 14, wherein the means for generating executable code further includes, where the specification is too complex to generate executable code directly, means for refining the method by specifying a series of operations to be undertaken, to produce a refined method and for verifying that the refined method

behaves in accordance with the abstract specification before generating executable code.

16. (original) The system as claimed in claim 14, wherein the means for running and evaluating the executable code, includes the further means, where the code does not meet requirements, for identifying the defects and for correcting the abstract specification and then for repeating checking the abstract specification for errors and verifying that the class has the expected properties.

17. (original) A system as claimed in claim 14, wherein he means for evaluating the speed of performance, further includes, where the speed of performance is inadequate, means for restructuring the data maintained by the class, adding variable declarations for the restructured data, and refining the methods and constructors to take account of the restructured data and generating executable code.

18. (original) A system as claimed in claim 14, wherein the means for evaluating the speed of performance, further includes, where the speed of performance is inadequate, means for refining the method to produce a refined method and verifying that the refined method meets the specification of the original method, and generating executable code.

19. (original) A system as claimed in claim 14, wherein the abstract specification includes a name of the class, a list of other class or classes and/or interfaces that said class inherits from, an abstract model of data maintained by the class, abstract specification of the methods and constructors of the class, and theorems of expected behaviour of the class and

the methods and constructors.

20. (original) A system as claimed in claim 19, wherein the abstract model of data includes declarations of abstract variables and may optionally include class invariants which are conditions concerning values of the abstract variables that are expected always to be true and declarations of abstract methods private to the class that assist in defining the class invariants and what other methods achieve.

21. (original) A system as claimed in claim 19, wherein the abstract specifications of the methods and constructors of the class include: a method name; a method parameter list; and a definition of what the method achieves.

22. (original) A system as claimed in claim 21, wherein the abstract specifications of the methods and constructors of the class further include one or more of: a method result type; a precondition that is required to hold whenever the method is called; a post-assertion description of conditions expected to hold when the method returns.

23. (original) A system as claimed in claim 15, wherein the means for refining the method includes means for providing an algorithm of program statements including postcondition statements.

24. (original) A statement as claimed in claim 19, wherein a description of the class is divided into regions; an abstract region containing the abstract data model and also private

methods and constructors referred to elsewhere in specifications of the class; an internal region containing re-implemented data and redundant data and also private methods and constructions referred to in re-implementations of other methods; a confined region of methods and constructors that are used only within the class and within other classes that inherit from that class; and an interface region of methods and constructors accessible to any program or components that uses instances of the class.

25. (original) A system as claimed in claim 14, wherein the means for generating executable code includes means for:

xii) tokenizing the abstract specification to form a token stream and building a presentation of the abstract specification;

xiii) passing the token stream;

xiv) binding identifiers and operators to declarations;

xv) converting the specification and expressions contained therein to standard forms;

xvi) generating a new instance of each variable at every point which the variable is changed, effectively replacing each variable by a succession of constants;

xvii) generating proof obligations represent requirements for program correctness;

xviii) proving the obligations;

xix) using the abstract specification to generate a test harness or a set of test data for testing speed of performance;

xx) generating code statements to implement the specification where no code is provided;

xxi) translating the code statements to easily translatable form; and

xxii) translating the easily translatable form to an output language.

26. (canceled)